EBNF Pseudo Code Syntax

This document defines the syntax and features of the Pseudo language used for Introduction to Programming class.

1. Lexical

Syntax:

```
Letter      :=  'a'-'z' | 'A'-'Z'

Digit       :=  '0'-'9'

Name        := Letter (Letter | Digit | '_' )*

String      := '"'  < any characters but double quote > '"'

Character   := '\'' < any character but single quote > '\''

Number      :='.'   Digit+ | ( Digit+ [ '.' Digit+])

Type        :=  "Real" | "Integer" | "String" | "Boolean" | "Character"

Literal     := Number | String | Character

BoolValue   := "True" | "False"

Comment     := "//" < any characters but \n > '\n'
```

2. Assert

A run-time test to verify correct operation. Can display a message like a Display.

Syntax:

```
Assert  :=  Expression "IfNot" Expression+
```

Examples

```
Set x    = 100
Set flag = True
Assert x==100 IfNot "Failed x, x=",x
Assert flag   IfNot "Failed flag true"
```

3. Bool Expression

*A **Boolean** expression can be only **True** or **False** and includes the comparison operators and the logical **AND**, **OR** operators.*

Syntax:

```
BoolExpression  := BoolValue                                  |
                   Expression CompareOperator Expression      |
                   BoolExpression "OR" BoolExpression         |
                   BoolExpression "AND" BoolExpression        |
                   "NOT" BoolExpression

CompareOperator := "<" | "<=" | "==" | ">=" | ">" | "!="
```

Examples:

```
"Bob" > "Ted"
True
A > B AND (C <= D)
```

4. Call

*Transfer control to a **Module** with optional arguments that match the Modules parameters by **Type**.*

Syntax:

```
CallStatement := "Call" '(' Argument* ')'

Argument  := Name | Expression
```

Examples

```
        Call  getWeight(weight)
        Call  getHeight(height)
        Call  showBMI(weight, height)
```

5. Class

Defines a description of an object with methods and fields.

Syntax:

```
ClassStatement := "Class" Name
                    (MethodStatement | FunctionStatement    | FieldStatement |
ConstantStatement)*
                  "End" "Class"
FieldStatement    := ("Private" | "Public" ) DeclareStatement
MethodStatement   := ("Private" | "Public" ) (FunctionStatement | ModuleStatement)
```

Examples:

```
Class CellPhone
      // Field declarations
      Private String manufacturer
      Private String modelNumber
      Private Real retailPrice
      // Method definitions
      Public Module setManufacturer(String manufact)
        Set manufacturer = manufact
      End Module
      Public Module setModelNumber(String modNum)
        Set modelNumber = modNum
      End Module
      Public Module setRetailPrice(Real retail)
        Set retailPrice = retail
      End Module
      Public Function String getManufacturer()
        Return manufacturer
      End Function
      Public Function String getModelNumber()
        Return modelNumber
      End Function
      Public Function Real getRetailPrice()
        Return retailPrice
      End Function
End Class
```

6. Constant

Used to create a constant variable that cannot be changed.

Syntax:

```
ConstantStatement := "Constant" Type Name '=' Expression
```

Examples:

```
Constant Integer SECONDS_IN_MINUTE = 60
Constant String Title = "Test of Time"
```

7. Declare

Used to create variables of a specific type.

Syntax:

```
DeclareStatement  := "Declare" TypeClause (',' TypeClause)*
TypeClause        := Type NameClause (',' SimpleClause | ArrayClause)*
SimpleClause      := Name ['=' Expression ]
ArrayClause       := Name'[' Number ']' ['=' Expression (',' Expression)*
```

Examples:

```
Declare Real Seconds, BMI
Declare Integer Weight = 100, Real Speed = 10.0, String Name = "Test of Time"
Constant Integer SIZE = 5
Declare Integer numbers[SIZE] = 5, 10, 15, 20, 25
```

8. Display

Used to output to the console.

Syntax:

```
DisplayStatement  := "Display" Argument (',' Argument)*
Argument          := Name | Expression
```

Examples

```
Display "This is a message"
Display MyVariable, MyOtherVariable, 10, "Test Message"
Display 10*56+1
Display "Your weight is optimal."
```

9. Do-Until

Loop over statement(s) until the **BoolExpression** *is true (continue while it is false). Executes statements at least once.*

Syntax:

```
DoUntilStatement := "Do"
                        Statement*
                    "Until" BoolExpression
```

Examples

```
Set count = 10
Do
       Display count
       Set count = count - 1
Until count == 0
```

10. Do-While

Loop over statement(s) while the **BoolExpression** *is true (continue while it is true). Executes statements at least once.*

Syntax:

```
DoWhiletatement := "Do"
                        Statement*
                    "While" BoolExpression
```

Examples

```
Set count = 10
Do
       Display count
       Set count = count - 1
While count > 0
```

11. ExpectFail

A run-time testing structure to trap run-time errors and report that the error occurred.

Syntax:

ExpectFail := **"ExpectFail"** ***Statement+*** **"End ExpectFail"**

Examples

```
 Module main()
    Call Test(300)
    ExpectFail
        Call Test(3.3) // expects an integer, loss of precision with real
    End ExpectFail
End Module

Module Test(Integer v)
    Display "Result is ",v
End Module
```

12. Expression

*A **mathematical** expression can be a variable **Name**, a **Literal** or a binary operation or a unary operation. Precidence is controlled by parentheses.*

Syntax:

```
Expression      := Name
                   Name  '[' Expression ']' ['[' Expression ']']                |
                   Literal                                                      |
                   Expression ('+' | '-' | '*' | '/' | '^' | "MOD") Expression  |
                   '+' Expression                                               |
                   '-' Expression                                               |
                   '(' Expression ')'
```

Examples:

```
"Test Message"
Declare Real Weight=1, Real Height = 100, Real duration= -5, Real Array[50],Integer x=1, Integer
Array2[5][3]
(Weight+100)*10 / (Height ^ 2)
+10 - ( -duration)
Array[x] *10
Array2[1][x] + 100
```

13. For

Loops over a range of values defined by an initial expression and a final expression. Each time the statements are evaluated with the variable Name, having initially the starting value, then incrementing by 1, and the statements execute again. This loops until the value of Name is equal the final expression.

Syntax:

```
ForStatement  := "For" Name '=' Expression "To" Expression [ "Step" Expression]
                       Statement*
                  "End" "For"
```

Examples

```
Declare Integer counter
For counter = 1 To 10
Display "Hello world: ",counter
End For
```

14. ForEach

Loops over all members of an array, assigning the Name the value of each member of the array and executing the Statements in the body. The loop may have an optional step to add to the counter on each iteration. A negative step causes the counter to run backwards, initial value is greater than the Test expression (final value).

Syntax:

```
ForStatement  := "ForEach" Name '=' ArrayName
                       Statement*
                  "End" "ForEach"

ArrayName     := Name
```

Examples

```
      Constant Integer SIZE = 5
      Declare Integer numbers[SIZE] = 5, 10, 15, 20, 25
      Declare Integer num
      For Each num In numbers
            Display num
      End ForEach
```

15. Function

*Transfer control to a **Function** with optional arguments that match the **Functions** parameters by **Type**. A value is returned by the **Return** statement in the function and the returned type is defined in the Function declaration.*

Syntax:

```
FunctionStatement := "Function" Type Name ''('' Parameter (',' Parameter)* ')'
                         (Statement | ReturnStatement)*
                     "End" "Function"
Parameter         := Type ["Ref"] Name
ReturnStatement   := "Return" Expression
```

Examples

```
Function Integer sum(Integer num1, Integer num2)
	Declare Integer result
	Set result = num1 + num2
	Return result
End Function
```

16. If

*Transfer control to a **Module** with optional arguments that match the Modules parameters by **Type**.*

Syntax:

```
IfStatement      := "If" BoolExpression "Then"
                         Statement+
                    ["Else" ["If" BoolExpression "Then"] Statement+ ]
                    "End" "If"
```

Examples

```
// If Form
If sales > 50000 Then
       Set bonus = 500.0
       Set commissionRate = 0.12
       Display "You've met your sales quota!"
End If

// If-Else form
If temperature < 40 Then
       Display "A little cold, isn't it?"
Else
       Display "Nice weather we're having."
End If
// If-Else-If form
If x < 10 Then
       Display "x is less than 10"
Else If x < 20 Then
       Display "x is less than 20 and greater than or equal to 10"
Else
       Display "x is greater than or equal to 20"
End If
```

17. Input

Used to get input from the console.

Syntax:

InputStatement **:=** **"Input"** ***Name*** (',' ***Name***)*

Examples

```
Input MyVariable1, MyVariable2
Input Weight
```

18. Module

Used to define a computational unit or a component of a program. ***Parameter*** *defines a* ***Type*** *optional* ***Ref*** *and the* ***Name*** *of the variable. Separated by commas.*

Syntax:

ModuleStatement **:=** **"Module"** **'('** ***Parameter*** **(','** ***Parameter*****)*** **')'** ***Statement**** **"End"** **"Module"**

Parameter **:=** ***Type*** **["Ref"]** ***Name***

Examples

```
Module getWeight (Real Ref weight)
	Display "Enter your weight in pounds: "
	Input weight
End Module
Module showTime (Integer Seconds, Integer Days, Integer Hours, Integer Minutes)
	Display "Days: ", Days
	Display "Hours: ", Hours
	Display "Minutes: ", Minutes
	Display "Seconds: ", Seconds
End Module
```

19. Program

A list of statements to execute.

Syntax:

```
Program          := ["Program"]  Statement* ["End" "Program"]

Statement        := DeclareStatement     |
                    InputStatement       |
                    DisplayStatement     |
                    SetStatement         |
                    ModuleStatement      |
                    CallStatement        |
                    IfStatement          |
                    SelectStatement      |
                    WhileStatement       |
                    DoWhileStatement     |
                    DoUntilStatement     |
                    ForStatement         |
                    ForEachStatement     |
                    ClassStatement       |
                    FunctionStatement
```

Examples:

```
Program
      Display "Hello World"
End Program


// without Program syntax


Display "Hello World"
```

20. Select

Allows the selection of code to execute based on a set of matching values along with a default (nothing matches option).

Syntax:

```
SelectStatement := "Select" Expression
                       (CaseStatement | DefaultStatement)*
                    "End" "Select"

CaseStatement     := "Case" Literal ':' Statement*

DefaultStatement  := "Default" ':' Statements*
```

Examples

```
Select month
      Case 1:
            Display "January"
      Case 2:
            Display "February"
      Case 3:
            Display "March"
      Default:
            Display "Error: Invalid month"
End Select
```

21. Set

Assign a value to a variable. The variable must be declared before it can be assigned.

Syntax:

```
SetStatement      := "Set" Name '=' Expression
```

Examples

```
Set weight = 150
Set MyName = "Grace Hopper"
Set BMI = weight * (703 / height ^ 2)
```

22. While

Loop over statement(s) while the **BoolExpression** *is true (continue while it is true). Tests each time before statements are executed.*

Syntax:

```
WhileStatement := "While" BoolExpression
                        Statement*
                  "End" "While"
```

Examples

```
Set count = 0
While count < 10
      Display count
      Set count = count + 1
End While
```